Nathaniel Serrano

COS 420

Program 3

2.12.2025

The previous two assignments did a good job of highlighting the value that AI tools can bring to the overall workflow of programming. This short comparison will examine this value by describing the difference of completing the same task with and without such tools. To start, it is important to analyze both the quality of the code written, as well as the time it took to write that code and complete each assignment. While neither of the assignments were timed this time around, they will be in the future. For now, analysis will be done via estimates of time it took to complete each task. Program 1, which was conducted without the use of AI tools, took approximately three hours to complete from start to finish. This includes coding, testing, debugging, and running the program twice with different parameters. Program 2, done with AI tools, was completed in around 30-45 minutes, including the process of prompting and generating the code, testing for bugs, and creating a report on the assignment.

Clearly, one method was much quicker. Now, for the obvious question: is the quality of code written similar? To that, the answer is a resounding yes. Testing took a bit longer with the AI tool's output than what was handwritten because of the fact that there were elements that I tested as I coded and knew worked with the end result, while for the AI tool's output, I had to test every element of its program to ensure everything worked. That being said, testing the AI's output did not lead to the discovery of any bugs outside of the lack of WimpPlayer being included in the final product, something that was easily fixed with an additional prompt. Testing

my own code, on the other hand, led to the discovery of a bug with the score tracker that took more time to resolve than what was spent generating the fully working code with the AI tool. It is important to note that though documentation was not required for Program 2 with the AI tool, I would have had to manually add documentation by hand if I wanted it to match an unorthodox standard like the one set in Player.java and WimpPlayer.java. AI tools as of now typically write code and documentation in a standard style, and it can be difficult to have them change that style consistently.

As discussed previously, Program 1's handwritten approach showed more bugs on the first try than the AI generated code of Program 2. That being said, debugging handwritten code can be a lot more independent over debugging code from an AI tool. Fortunately, Program 2 did not produce any noteworthy bugs this time around, but if it had, the further use of the AI tool to help with debugging would likely be required, since the user would not have the comprehensive understanding of the code that they get from writing it themselves. So while Program 2's approach may appear to be quicker and lead to less bugs, it's much more reliant on the continued use of AI tools in the event that bugs do occur. In addition, it is vital to note that the reason AI tools worked so well with this assignment is due to the simplicity of the tasks, as well as the detailed requirements outlined in the assignment on BrightSpace. If AI tools were instead used on an assignment that involved working on a large codebase like an operating system, writing programs that worked and were integrated seamlessly into the codebase would be a far more difficult endeavor for the AI tool on its own. In fact, it is likely that the AI would struggle to accomplish this task without at least producing bugs and requiring the user to step in.

These AI tools have taken the world by storm in recent years, bringing with them change that programmers must acclimate to if they wish to succeed in this new era of programming.

These tools are far from perfect but bring with them an invaluable, more efficient workflow that can lead to far more productivity. I'm curious to see how they will continue to evolve in the future.